Towards an Empirical Foundation for Assessing Bayesian Optimization of Hyperparameters

Katharina Eggensperger, Matthias Feurer, Frank Hutter
Freiburg University
{eggenspk,feurerm,fh}@informatik.uni-freiburg.de

James Bergstra
University of Waterloo
james.bergstra@uwaterloo.ca

Jasper Snoek
Harvard University
jsnoek@seas.harvard.edu

Holger H. Hoos and Kevin Leyton-Brown
University of British Columbia
{hoos,kevinlb}@cs.ubc.ca

Abstract

Progress in practical Bayesian optimization is hampered by the fact that the only available standard benchmarks are artificial test functions that are not representative of practical applications. To alleviate this problem, we introduce a library of benchmarks from the prominent application of hyperparameter optimization and use it to compare Spearmint, TPE, and SMAC, three recent Bayesian optimization methods for hyperparameter optimization.

1 Introduction

The performance of many machine learning (ML) methods depends crucially on hyperparameter settings and thus on the method used to set hyperparameters. Recently, Bayesian optimization methods have been shown to outperform established methods for this problem (such as grid search and random search [1]) and to rival—and in some cases surpass—human domain experts in finding good hyperparameter settings [2, 3, 4]. As a result, hyperparameter optimization has become an active research area within Bayesian optimization, with characteristics such as low effective dimensionality [1, 5, 6] and problem variants, such as optimization across different data sets [7] being explored.

One obstacle to further progress in this nascent field is a dearth of hyperparameter optimization benchmarks and comparative empirical studies. It can be difficult to evaluate a new optimizer on benchmarks used in previous papers because (1) optimizers are written in different programming languages and use different search space representations and file formats; (2) hyperparameter optimization benchmarks that have been developed jointly with an optimizer are not typically packaged as black boxes (including the respective machine learning algorithm and its input data) that can be used with other optimizers.

To alleviate these problems, we have collected and made available a library of hyperparameter optimization benchmarks from the recent literature and used it to empirically evaluate the respective strengths and weaknesses of three prominent Bayesian optimization methods for hyperparameter optimization: SPEARMINT [2], TPE [8], and SMAC [9]. We thereby hope to provide an empirical foundation to facilitate the development and evaluation of future methods for this problem.

2 Bayesian Optimization Methods for Hyperparameter Optimization

Given a machine learning algorithm A having hyperparameters $\lambda_1, \dots, \lambda_n$ with respective domains $\Lambda_1, \dots, \Lambda_n$, we define its hyperparameter space $\mathbf{\Lambda} = \Lambda_1 \times \cdots \times \Lambda_n$. For each hyperparameter setting $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$, we use $A_{\boldsymbol{\lambda}}$ to denote the learning algorithm A using this setting. We further use $\mathcal{L}(A_{\boldsymbol{\lambda}}, \mathcal{D}_{\text{train}}, \mathcal{D}_{\text{valid}})$ to denote the validation loss (e.g., misclassification rate) that $A_{\boldsymbol{\lambda}}$ achieves on data $\mathcal{D}_{\text{valid}}$ when trained on $\mathcal{D}_{\text{train}}$. The hyperparameter optimization problem under k-fold cross-validation is then to minimize the blackbox function

$$f(\boldsymbol{\lambda}) = \frac{1}{k} \sum_{i=1}^{k} \mathcal{L}(A_{\boldsymbol{\lambda}}, \mathcal{D}_{\text{train}}^{(i)}, \mathcal{D}_{\text{valid}}^{(i)}). \tag{1}$$

We note that an alternative to optimizing hyperparameters is to marginalize over them in a Bayesian model averaging framework; however, in most cases the costs of doing so are prohibitive, and we therefore do not consider that alternative here.

Hyperparameters can be continuous, integer-valued, or categorical. Following [10] and [8], we say that a hyperparameter λ_i is *conditional* on another hyperparameter λ_j if λ_i is only active if hyperparameter λ_j takes values from a given set $V_i(j) \subsetneq \Lambda_j$. These conditional parameters are, for example, common in deep architectures, or in frameworks including many alternative algorithms, and some hyperparameter optimizers exploit knowledge about these conditionalities in their models to improve their performance [9, 8, 11, 12].

Bayesian optimization (see [13] for a detailed tutorial) constructs a probabilistic model $\mathcal{M}$ of f based on point evaluations of f and any available prior information, and uses that model to select subsequent configurations $\boldsymbol{\lambda}$ to evaluate. In order to select its next hyperparameter configuration $\boldsymbol{\lambda}$ using model $\mathcal{M}$, Bayesian optimization uses an *acquisition function* $a_{\mathcal{M}} : \boldsymbol{\Lambda} \to \mathbb{R}$, which uses the predictive distribution of model $\mathcal{M}$ at arbitrary hyperparameter configurations $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$ to quantify how useful knowledge about $\boldsymbol{\lambda}$ would be. This function is then maximized over $\boldsymbol{\Lambda}$ to select the most useful configuration $\boldsymbol{\lambda}$ to evaluate next. Several well-studied acquisition functions exist [14, 15, 16]; all aim to trade off exploitation (locally optimizing hyperparameters in regions known to perform well) versus exploration (trying hyperparameters in a relatively unexplored region of the space). The most popular acquisition function is the *expected improvement* [15] over the best previously-observed function value f_{min} attainable at a hyperparameter configuration $\boldsymbol{\lambda}$ (where expectations are taken over predictions with the current model $\mathcal{M}$):

$$\mathbb{E}_{\mathcal{M}}[I_{f_{min}}(\boldsymbol{\lambda})] = \int_{-\infty}^{f_{min}} \max\{f_{min} - f, 0\} \cdot p_{\mathcal{M}}(f \mid \boldsymbol{\lambda}) \, df. \tag{2}$$

One main difference between existing Bayesian optimization algorithms lies in the model classes they employ. In this paper, we empirically compare three popular Bayesian optimization algorithms for hyperparameter optimization that are based on different model types.

SPEARMINT [2, 17]. SPEARMINT uses a Gaussian process (GP) to model $p_{\mathcal{M}}(f \mid \boldsymbol{\lambda})$ and performs slice sampling over the GP's hyperparameters [18]. It supports continuous and discrete parameters (by rounding), but does not provide a mechanism to exploit knowledge about conditional parameters.

Sequential Model-based Algorithm Configuration (SMAC) [9, 19]. SMAC uses random forests to model $p_{\mathcal{M}}(f \mid \boldsymbol{\lambda})$ as a Gaussian distribution whose mean and variance are the empirical mean and variance over the predictions of the forest's trees. For hyperparameter optimization problems with cross-validation, SMAC evaluates the loss of configurations at single folds at a time in order to save time. Different configurations are compared based only on the folds evaluated for both. SMAC supports continuous, categorical, and conditional parameters. It was the best-performing optimizer for Auto-WEKA [3] and has also been used to configure many combinatorial optimization algorithms.

Tree Parzen Estimator (TPE) [8, 20]. TPE is a non-standard Bayesian optimization algorithm. While SPEARMINT and SMAC model $p(f \mid \lambda)$ directly, TPE models $p(f < f*)$, $p(\boldsymbol{\lambda} \mid f < f*)$, and $p(\boldsymbol{\lambda} \mid f \geq f*)$, where $f*$ is defined as a fixed quantile of the losses observed so far, and the latter two probabilities are defined by tree-structured Parzen density estimators. With these distributions defined, a term proportional to the expected improvement from Equation (2) can be computed in closed form [8]. TPE supports continuous, categorical, and conditional parameters, as well as priors for each hyperparameter over which values are expected to perform best. It has been used succesfully in several papers beyond the one in which it was introduced [4, 21, 3].

Table 1: Hyperparameter optimization benchmarks used. The loss function is the function value for Branin and Hartmann 6d, perplexity for LDA and misclassification rate for all the others. The runtime column gives the 0.1 and 0.9 quantiles over all function evaluations performed by all optimizers, in minutes. For benchmarks with crossvalidation these are the runtimes for one fold.

Algorithm	**#hyp.params (conditional)**	**continuous/ discrete**	**Dataset**	**Size (Train/Valid/Test)**	**Citation**	**Runtime [min] q0.1 / q0.9**
Branin	2(-)	2/-	-	-	[22]	trivial
Hartmann 6d	6(-)	6/-	-	-	[22]	trivial
Log. Reg.	4(-)	4/-	MNIST	50k/10k/10k	[2, 23]	0.3/12.5
LDA ongrid	3(-)	-/3	Wikipedia articles	200k/24560/25k	[2, 24]	table lookup
SVM ongrid	3(-)	-/3	UniPROBE	≈ 20k/-/≈ 20k	[2, 25]	table lookup
HP-NNET	14(4)	7/7	MRBI	10k/2k/50k	[8, 26]	0.9/13.4
HP-NNET	14(4)	7/7	convex	6.5k/1.5k/50k	[8, 26]	0.7/16.7
HP-DBNET	38(29)	19/17	convex	6.5k/1.5k/50k	[8, 26]	0.7/46.3
Auto-WEKA	786(784)	296/490	convex	10 fold cv, 8k Train, 50k Test	[27, 3, 26]	0.4/15.2
Log. Reg. 5CV	4(-)	4/-	MNIST	5 fold cv, 60k Train, 10k Test	[2, 23]	0.3/12.1
HP-NNET 5CV	14(4)	7/7	MRBI	5 fold cv, 12k Train, 50k Test	[8, 26]	0.9/10.8
HP-NNET 5CV	14(4)	7/7	convex	5 fold cv, 8k Train, 50k Test	[8, 26]	0.7/10.7

3 Hyperparameter Optimization Benchmarks

Table 1 summarizes all of the benchmarks we collected for the first version of our *hyperparameter optimization library, HPOlib*, which is available at `www.automl.org/hpolib/`. HPOlib includes simple test functions (used for convenience in many papers) as well as the following benchmarks:

Low-dimensional benchmarks. We collected three benchmarks with few parameters from [2]: simple LOGISTIC REGRESSION to classify the popular MNIST dataset; ONLINE LATENT DIRICHLET ALLOCATION (LDA) for Wikipedia articles, and STRUCTURED SUPPORT VECTOR MACHINES (SVM). The latter two benchmarks are defined on a grid of hyperparameter values: for each of the grid points (288 for LDA; 1400 for SVM), algorithm performance data has been precomputed by [2] to allow for very rapid experiments. Another advantage of such precomputed data is that anyone can use these benchmarks without having to compile and run the respective ML algorithms.

Medium-dimensional benchmarks. We collected two types of benchmarks of intermediate dimensionality from [8]. HP-NNET and HP-DBNET are implementations of a simple neural network and a deep neural network, respectively. Both run faster on GPUs than CPUs, and both include continuous and categorical parameters, some of which are conditional. For each hyperparameter in these benchmarks, an expert-defined prior over good values is defined. Since these priors cannot be expressed in SMAC's and SPEARMINT's formats, they get lost in translation, as does conditionality information in the case of SPEARMINT.

High-dimensional benchmarks. To test the limits of current optimizers, we also investigated the AUTO-WEKA framework [3], which encodes combined model selection and hyperparameter optimization into an enormous hierarchical (i.e., highly conditional) space with 768 hyperparameters.

All benchmarks in HPOlib can be called through the command line. This both allows for the support of optimizers written in arbitrary programming languages and allows us to control their use of resources. HPOlib includes python scripts that offer a common interface to the three Bayesian optimization algorithms used throughout this paper (and can convert between the optimizers' different input formats). These scripts call the optimizers and wrap their calls to the machine learning algorithm being optimized with a tool called runsolver [28] to ensure that they respect given time and memory limits. If they do not do so (consider, e.g., a call to a neural network setting the number of layers to 1 000), they are terminated (using SIGTERM, 200 seconds grace period, and then SIGKILL; this process allows partially trained models to return the quality of their current model). Terminated or otherwise crashed calls to the machine learning algorithm without valid output yield the worst possible result for the hyperparameter setting being evaluated. All calls to the machine learner and their results are stored in a uniform optimizer-independent format, to facilitate the analysis of results. Finally, the HPOlib scripts also allow restarting interrupted optimizer runs from their last state.

Since overfitting is a critical issue in hyperparameter optimization (especially as hyperparameter optimization methods improve), HPOlib supports k-fold cross-validation, either by evaluating all k folds at once, or by evaluating one fold at a time. Out of the benchmarks above only Auto-WEKA used cross-validation, so to study the importance of cross-validation we added additional versions of the LOGISTIC REGRESSION and HP-NNET benchmarks with 5-fold cross-validation.

Table 2: Losses obtained for all optimizers and benchmarks . We report means and standard deviation across 10 runs of each optimizer. For each benchmark, bold face indicates the best mean loss, and underlined values are not statistically significantly different from the best according to an unpaired t-test (with p=0.05). For HP-NNET we also provide results for half the function evaluation budget to quantify the improvement over time.

		SMAC		Spearmint		TPE	
Experiment	#evals	Valid. loss	Best loss	Valid. loss	Best loss	Valid. loss	Best loss
branin (0.398)	200	0.655±0.27	0.408	**0.398**±0.00	**0.398**	0.526± 0.13	0.422
har6 (-3.322)	200	-2.977±0.11	-3.154	**-3.133**±0.41	**-3.322**	-2.823±0.18	-3.039
Log.Regression	100	8.6±0.9	7.7	**7.3**±0.2	**7.0**	8.2±0.6	7.5
LDA ongrid	50	**1269.6**±2.9	**1266.2**	1272.6±10.3	**1266.2**	1271.5±3.5	**1266.2**
SVM ongrid	100	**24.1**±0.1	**24.1**	24.6±0.9	**24.1**	24.2±0.0	**24.1**
HP-NNET convex	100	**19.5**±1.5	**17.0**	20.6±0.3	20.1	**19.5**±1.6	17.4
HP-NNET convex	200	**18.3**±1.9	**15.2**	20.0±0.9	17.3	18.5±1.4	16.2
HP-NNET MRBI	100	51.5±2.8	**46.1**	52.2±3.3	46.5	**50.0**±1.7	47.3
HP-NNET MRBI	200	**48.3**±1.80	**46.1**	51.4±3.2	46.5	48.9±1.4	46.9
HP-DBNET convex	100	**16.4**±1.2	**14.5**	20.74±6.9	15.5	17.29±1.7	15.3
HP-DBNET convex	200	**15.4**±0.8	**14.0**	17.45±5.6	14.6	16.1±0.5	15.3
Auto-WEKA	30h	**27.5**±4.9	**22.3**	40.64±7.2	31.9	35.5±2.9	28.8
Log.Regression 5CV	500 folds	**8.1**±0.2	**7.8**	8.2±0.1	7.9	8.9±0.5	8.1
HP-NNET convex 5CV	500 folds	**18.2**±1.5	**16.9**	23.0±5.0	19.7	20.9±1.3	18.6
HP-NNET MRBI 5CV	500 folds	**47.9**±0.7	47.2	52.8±5.1$^{(9)}$	**46.6**	50.8 ±1.4	48.2

4 Experiments

We ran each optimizer with its default settings 10 times on each benchmark for the number of function evaluations used in the paper introducing the benchmark. We used a runsolver timeout of one hour for individual runs (in case of cross-validation, for individual folds); this only took effect for a few runs on the HP-NNET and HP-DBNET experiments. The HP-NNET and HP-DBNET experiments were run on a cluster of NVIDIA Tesla M2070s GPUs, which imposed a wall time limit of 24 hours per optimizer run. Since this did not suffice to perform a sufficient number of function evaluations, we used HPOlib's scripts to restart the optimizers from their last saved state until the target number of function evaluations was achieved.

Table 2 summarizes our results. Overall, SPEARMINT performed best for the low-dimensional continuous problems.[1] For the higher-dimensional problems, which also include conditional parameters, SMAC and TPE performed better. However, we note that based on the small number of 10 runs per optimizer, many differences are not statistically significant.

For benchmarks including k-fold cross-validation, SMAC evaluated one fold at a time while the other methods (which do not yet support single fold evaluations) evaluated all k folds. SMAC thus managed to consider roughly 4 times more configurations than either TPE or SPEARMINT in the same budget of fold evaluations. Since the budget for each optimizer was expressed as a number of function evaluations, they were not penalized for choosing costly-to-evaluate hyperparameters, and in our experiments, SPEARMINT runs were sometimes up to a factor of three slower than those of TPE; in the future, we plan to use time budgets and SPEARMINT's time-sensitive EI criterion [2]. We also studied the CPU time required by the optimizers. SMAC's and TPE's overhead was negligible (< 1 second), but due to the cubic scaling behaviour of its GPs, SPEARMINT required > 42 seconds to select the next data point after 200 evaluations. This would prohibit its use for the optimization of cheap functions, but here this overhead was dominated by the expensive function evaluations.

5 Conclusion and Future Work

This work introduces a benchmark library for hyperparameter optimization and provides the first extensive comparison of three optimizers. To support further research, our software package and benchmarks are publicly available at `www.automl.org/hpolib/`. It offers a common interface for the three optimization packages utilized in this paper and allows the easy integration of new ones. Our benchmark library is only a first step, but we are committed to making it easy for other researchers to use it and to contribute their own benchmarks and optimization packages.

[1]However, it showed some robustness problems, e.g., crashing on 1 of 10 runs for the HARTMANN-6 function because of a singular covariance matrix. It also had problems with discrete parameter values: by maximizing expected improvement over a dense Sobol grid instead of the discrete input grid, it sometimes repeatedly chose values that were rounded to the same discrete values for evaluation (leading to repeated samples).

References

[1] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *JMLR*, 13:281–305, 2012.

[2] J. Snoek, H. Larochelle, and R.P. Adams. Practical Bayesian optimization of machine learning algorithms. In *Proc. of NIPS'12*, 2012.

[3] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *Proc. of KDD'13*, 2013.

[4] J. Bergstra, D. Yamins, and D. D. Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *Proc. of ICML'12*, 2013.

[5] B. Chen, R.M. Castro, and A. Krause. Joint optimization and variable selection of high-dimensional Gaussian processes. In *Proc. of ICML'12*, 2012.

[6] Z. Wang, M. Zoghi, F. Hutter, D. Matheson, and N. de Freitas. Bayesian optimization in high dimensions via random embeddings. In *Proc. of IJCAI'13*, 2013.

[7] R. Bardenet, M. Brendel, B. Kégl, and M. Sebag. Collaborative hyperparameter tuning. In *Proc. of ICML'13*, 2013.

[8] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for Hyper-Parameter Optimization. In *Proc. of NIPS'11*, 2011.

[9] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proc. of LION-5*, 2011.

[10] F. Hutter, H.H. Hoos, K. Leyton-Brown, and T. Stützle. ParamILS: an automatic algorithm configuration framework. *JAIR*, 36(1):267–306, 2009.

[11] F. Hutter and M.A. Osborne. A kernel for hierarchical parameter spaces, 2013. arXiv:1310.5738.

[12] K. Swersky, D. Duvenaud, J. Snoek, F. Hutter, and M.A. Osborne. Raiders of the lost architecture: Kernels for Bayesian optimization in conditional parameter spaces. In *NIPS workshop on Bayesian Optimization in theory and practice (BayesOpt'13)*, 2013. Published online.

[13] E. Brochu, V.M. Cora, and N. de Freitas. A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. eprint arXiv:1012.2599, arXiv.org, December 2010.

[14] D. R. Jones, M. Schonlau, and W. J. Welch. Efficient global optimization of expensive black box functions. *Journal of Global Optimization*, 13:455–492, 1998.

[15] M. Schonlau, W. J. Welch, and D. R. Jones. Global versus local search in constrained optimization of computer models. In *New Developments and Applications in Experimental Design*, volume 34, pages 11–25. 1998.

[16] N. Srinivas, A. Krause, S. Kakade, and M. Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. In *Proc. of ICML'10*, 2010.

[17] J. Snoek. Spearmint source code. github version from 04/08/2013; URL: `https://github.com/JasperSnoek/spearmint`.

[18] I. Murray and R. P. Adams. Slice sampling covariance hyperparameters of latent Gaussian models. In *Proc. of NIPS'10*, pages 1723–1731. 2010.

[19] S. Ramage and F. Hutter. SMAC source code, java version v2.06.01; URL: `www.cs.ubc.ca/labs/beta/Projects/SMAC/`.

[20] J Bergstra. TPE source code, part of hyperopt implementation, github version from 08/30/2013. URL: `https://github.com/hyperopt/hyperopt`.

[21] J. Bergstra and D.D. Cox. Hyperparameter optimization and boosting for classifying facial expressions: How good can a "null" model be? *CoRR*, abs/1306.3476, 2013.

[22] A. Hedar. Test functions for unconstrained global optimization. `http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/TestGO_files/Page364.htm`.

[23] Y. Lecun, l. Bottou, Y. Bengio, and P.Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, volume 86, pages 2278–2324, 1998.

[24] M. D. Hoffman, D. M. Blei, and F. R. Bach. Online learning for latent dirichlet allocation. In *Proc. of NIPS'10*, 2010.

[25] K. Miller, M. P. Kumer, B. Packer, D. Goodman, and D. Koller. Max-margin min-entropy models. In *Proc. of AISTATS'12*, 2012.

[26] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio. An empirical evaluation of deep architectures on problems with many factors of variation. In *Proc. of ICML'07*, 2007.

[27] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I.H. Witten. The WEKA data mining software: an update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

[28] O. Roussel. Controlling a solver execution with the runsolver tool. *JSAT*, 7:139–144, 2011.